Identification of optimal contraction sequences for tensor networks

Robert N. C. Pfeifer[1, *]

[1] *Perimeter Institute for Theoretical Physics, 31 Caroline St. N, Waterloo ON N2L 2Y5, Canada*

(Dated: April 22, 2013)

Tensor network Ansätze provide powerful tools for the study of quantum many-body systems on a lattice in the low-energy regime, representing the state of a system as an efficiently-contractible network of multi-index tensors optimised numerically by means of a variational algorithm. The efficient contraction of tensor networks is vital to both the development and the implementation of tensor network algorithms, but determination of optimal contraction sequences is presently performed by human operators in a process which is both tedious and prone to error. This paper presents an algorithm for exhaustively searching the space of contraction sequences on a practical timescale, automating the determination of both the optimal contraction sequence and the cost function associated with the contraction of a given tensor network. By removing the burden of computing these contraction sequences from the researcher, this tool both facilitates the development of novel efficiently-contractible tensor network Ansätze and assists in the optimal implementation of existing tensor network algorithms.

PACS numbers: 02.70.–c, 05.30.–d, 03.67.–a

I. INTRODUCTION

Tensor network algorithms provide powerful tools for the study of a wide variety of physical systems. They are perhaps best known for their use in condensed matter physics as numerical techniques for the study of quantum many-body systems on a lattice [e.g. 1–24], but recent breakthroughs blending ideas from quantum information with advanced numerical techniques have led to the construction of new Ansätze [e.g. 25–28] having applications in fields as diverse as holography and the AdS/CFT correspondence [29–31] and the classification of topological phases in quantum spin systems [32–35]. As a numerical tool, a tensor network algorithm typically comprises an Ansatz for the description of pure or mixed quantum states, which is composed of a network of tensors, and an iterative procedure for updating this Ansatz. Examples include the Density Matrix Renormalisation Group (DMRG) [36, 37] and Time Evolving Block Decimation (TEBD) algorithms [38, 39], both of which are based on the Matrix Product State (MPS) Ansatz, and also Tree Tensor Networks (TTNs) [40], Projected Entangled Pair States (PEPS) [41, 42], and the Multi-scale Entanglement Renormalisation Ansatz (MERA) [25–28].

The fundamental challenge to the numerical study of quantum many-body systems on a lattice is that the number of degrees of freedom, and thus the computational cost associated with exact simulation, grows exponentially with the size of the system. To overcome this challenge, tensor network Ansätze replace the coefficients $c_{i_1 \ldots i_n}$ of a quantum state $|\psi\rangle$

$$|\psi\rangle = \sum_{i_1 \ldots i_n} c_{i_1 \ldots i_n} |i_1, \ldots, i_n\rangle \tag{1}$$

with a network of tensors whose dimensions are such that the number of coefficients required to describe the network exhibits a better scaling in n, the number of lattice sites, than does the number of coefficients $c_{i_1 \ldots i_n}$ in Eq. (1). Indeed, for many tensor networks this scaling in n is polynomial rather than exponential.

Given this reduction in the number of coefficients in the description, a tensor network Ansatz is capable only of representing states which lie within some restricted region of the Hilbert space of the system, but nevertheless these Ansätze and associated algorithms are capable of providing substantial insight into the physics of a wide variety of systems in appropriate regimes (again see Refs. 1–24 for examples).

In order for a tensor network algorithm to be useful as a tool for numerical computation, it must be possible to perform the operations of the algorithm for a reasonable computational cost. An economical description of the relevant part of the Hilbert space of a system is a good start, but this is not the only factor which must be taken into account: When determining whether a given tensor network algorithm is computationally feasible, the structure of the tensor network itself also plays a significant role. In describing scaling of the cost of a tensor network algorithm, it is customary to express this in the form of a polynomial in some refinement parameter χ, which may (for example) correspond to the dimensions of indices within the tensor network. Assuming that the coefficients of this cost polynomial are small, costs which scale as excessively large powers of χ may then reflect an algorithm which pushes the limits of computational feasibility.

The trade-off between sophistication of an Ansatz and the associated cost of the update algorithm represents a critical tension in the development of novel tensor network algorithms. For example, the structure of the 4:1 2D MERA [6, 43] indicates that this particular Ansatz will provide a powerful representation of highly-entangled

* rpfeifer@perimeterinstitute.ca

2D systems, and this has been confirmed analytically in Ref. 44 where it is shown to furnish a compact and physically meaningful description of the toric code. However, numerical computations using this Ansatz are hindered by an update cost of $O(\chi^{26})$. The ability to quickly and conveniently determine the cost of contracting different tensor networks is therefore of great importance to researchers employed in the development of novel tensor network algorithms.

Even when the cost of updating a tensor network algorithm is known, the implementation of these algorithms is frequently a non-trivial affair. The process of contracting a tensor network may always be viewed as a series of pairwise contractions, and the overall cost of contracting the network is highly dependent upon the sequence in which these contractions are carried out. For instance, the network shown in Fig. 1(i) is one network which must be contracted during the variational optimisation of the 3:1 1D MERA [27, 28]. The most efficient contraction sequence yields a cost of $O(\chi^8)$, but careless choices of sequence can yield costs as high as $O(\chi^{11})$. Thus the ability to determine the optimal contraction sequence for a tensor network is also important to those implementing pre-existing tensor network algorithms, if these are to be implemented in a computationally efficient manner.

Until now, the only option for determining the optimal contraction sequence for a tensor network algorithm has been a labour-intensive manual study of the tensor network, and for all but the simplest networks an exhaustive search is unfeasible. The process is time-consuming, and for networks composed of many tensors with similar numbers of indices but subtly different connectivity, it can be difficult to be certain whether any contraction sequence is indeed optimal. This paper introduces an algorithm which determines an optimal contraction sequence for any given tensor network, permitting researchers who design and implement tensor network algorithms to quickly and easily determine efficient network contractions and to identify with confidence the minimal computational cost. The algorithm is described in Sec. II, and a reference implementation is provided in the archive associated with this paper and documented in Sec. III.

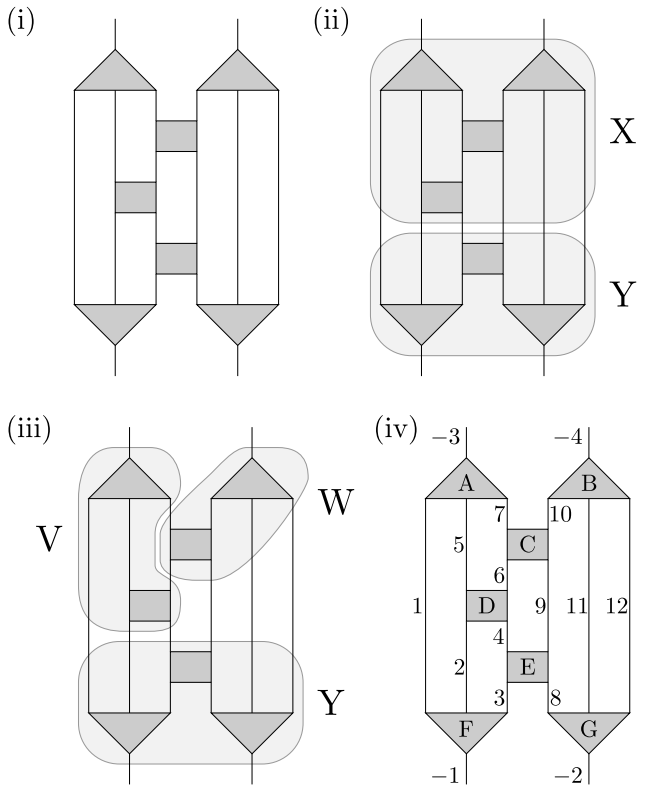


FIG. 1. (i) Graphical representation of one of the tensor networks which must be contracted during variational optimisation of the 3:1 MERA [27, 28]. (ii) A bipartition of this network defines two tensors, X and Y, corresponding to the contraction of the resulting sub-networks, as shown. A cost may be associated with this bipartition corresponding to the cost of contracting X with Y. (iii) The network associated with tensor X may then, in turn, be further subdivided into networks associated with tensors V and W, as shown. (iv) To describe bipartitions of a network, it is useful to label indices. In this diagram, summed indices are labelled with unique positive integers, and open indices are labelled with unique negative integers descending consecutively from −1. Tensors have also been labelled with letters to allow easy reference from the text.

II. ALGORITHM

This Section describes the Netcon algorithm (short for "Network Contraction") for performing an optimised search through the space of possible tensor network contraction sequences. The problem of determining the most efficient contraction sequence for a given tensor network is reformulated as a search through the space of sequential subdivisions of that network, first under the assumption that the initial network and all sub-networks are non-disjoint[1] and contain no traces (Sec. II A), and then allowing disjoint networks and sub-networks (Sec. II B). Additional techniques used to optimise the search process are discussed in Sec. II C, and finally traces are included in Sec. II D.

[1] A tensor network is disjoint iff its tensors can be separated into two sub-networks sharing no common indices. A tensor network is non-disjoint iff it is not disjoint.

A. Non-disjoint networks

1. Basic procedure

Consider a non-disjoint tensor network such as that shown in Fig. 1(i). This network may be contracted to a single tensor by evaluating the sums over paired indices in some sequence, with the computational cost of that sequence corresponding to the number of multiplication operations which must be performed. However, not all sequences of pairwise contractions attract equivalent computational costs. It is shown in Appendix A that for any tensor network there always exists a sequence which achieves the minimum possible cost using only pairwise contractions, and so the objective of the Netcon algorithm will be the identification of a pairwise contraction sequence which realises this minimum cost.

One might therefore propose an iterative algorithm to explore the space of possible contraction sequences, maintaining a list of available tensors and their connections and updating this list as the tensors undergo pairwise contraction. While this approach is viable, the coding (and performance) overhead associated with maintaining both the list of tensors and the history of individual tensors, so that composite tensors may be uncontracted and different contraction sequences explored, makes this approach unattractive.

The problem of determining the optimal contraction sequence for a tensor network such as Fig. 1(i) may be recast more favourably as follows: Rather than considering pairwise contractions of tensors, instead consider bipartitions of the original tensor network. An example bipartition is shown in Fig. 1(ii), and divides the original network (which will be denoted $\mathcal{N}_\mathrm{Z}$) into two sub-networks $\mathcal{N}_\mathrm{X}$ and $\mathcal{N}_\mathrm{Y}$ (which for now will also be assumed to be non-disjoint). The tensors obtained on contracting these two sub-networks may be denoted X and Y, and the minimum cost of contracting network $\mathcal{N}_\mathrm{Z}$ *finishing with the contraction of tensor X with tensor Y* is then the minimum cost of contracting network $\mathcal{N}_\mathrm{X}$, plus the minimum cost of contracting network $\mathcal{N}_\mathrm{Y}$, plus the cost associated with this final contraction. Provided the minimum costs for contracting all sub-networks such as $\mathcal{N}_\mathrm{X}$ and $\mathcal{N}_\mathrm{Y}$ are known, the minimum cost for contracting $\mathcal{N}_\mathrm{Z}$ may then be computed by iterating over all possible bipartitions of $\mathcal{N}_\mathrm{Z}$ and identifying the cheapest option. Evaluation of the costs to contract sub-networks such as $\mathcal{N}_\mathrm{X}$ and $\mathcal{N}_\mathrm{Y}$ may be performed by recursion as needed, with the cost associated with a trivial network (one containing only one tensor) being zero. If `getcost(`$\mathcal{N}_\mathrm{Z}$`)` is a function which performs the above-described iteration over bipartitions of $\mathcal{N}_\mathrm{Z}$, then `getcost(`$\mathcal{N}_\mathrm{Z}$`)` may compute the costs associated with contraction of networks $\mathcal{N}_\mathrm{X}$ and $\mathcal{N}_\mathrm{Y}$ by invoking `getcost(`$\mathcal{N}_\mathrm{X}$`)` and `getcost(`$\mathcal{N}_\mathrm{Y}$`)` respectively. In this way, a function `getcost()` may explore all possible contraction sequences for a network $\mathcal{N}_\mathrm{Z}$ without explicitly constructing the dimensions of the intermediate tensors.

In conjunction with this approach, it is useful to introduce a means of describing bipartitions of a tensor network which is similarly independent of the intermediate tensors involved. For a non-disjoint network partitioned into two non-disjoint sub-networks, any such partition may be uniquely described in terms of the summed index pairs which are split, such that one occurrence of the index lies on network $\mathcal{N}_\mathrm{X}$ and the other lies on network $\mathcal{N}_\mathrm{Y}$. In Fig. 1(iv) a unique label has been assigned to each index of Fig. 1(i); any convention may be adopted for these labels, but in the present paper all summed indices are labelled with positive integers and all unsummed indices with negative integers descending consecutively from -1. The tensors themselves have been labelled with the letters A... G for reference. Given the labelling of Fig. 1(iv), contraction of tensor X with tensor Y is associated with contraction over the indices labelled 1, 2, 4, 9, 11, and 12, and this list of indices is independent of how networks $\mathcal{N}_\mathrm{X}$ and $\mathcal{N}_\mathrm{Y}$ might be subdivided in turn. One may then subdivide network $\mathcal{N}_\mathrm{X}$ into two portions, $\mathcal{N}_\mathrm{V}$ and $\mathcal{N}_\mathrm{W}$, as shown in Fig. 1(iii), with contraction of V and W being associated with the indices 6 and 7. Given tensors V, W, and Y, the index sequence

```
6 7 1 2 4 9 11 12
```

then specifies that first V is to be contracted with W, then the resulting tensor (X) is to be contracted with Y. A sequence containing all summed index labels on a network $\mathcal{N}_\mathrm{Z}$ [e.g. 1 to 12 in Fig. 1(iv)] thus specifies a unique sequence of pairwise contractions whereby that network may be fully contracted into a single tensor.

This method of describing contraction sequences may be contrasted with one based on the letter labels of Fig. 1(iv): If brackets () are used to denote pairwise contractions, so that e.g. (BC) represents the contraction of tensor B with tensor C, then tensors X and Y can each be represented by any of a number of different sequences depending on the contraction sequence by which they are constructed, and the way in which contraction of X with Y is represented will vary accordingly. These two methods of describing contraction sequences are interchangeable for non-disjoint networks; for example, for Fig. 1(iv) the index sequence

```
9 4 6 5 7 2 1 3 11 12 8 10
```

uniquely specifies the pairwise contraction sequence

(((((CE)D)A)F)(BG))

and vice versa. However, within the Netcon algorithm an index-based approach is preferable as the description of a bifurcation of network $\mathcal{N}_\mathrm{Z}$ is then independent of the optimal contraction sequences of the sub-networks $\mathcal{N}_\mathrm{X}$ and $\mathcal{N}_\mathrm{Y}$. Once an optimal sequence has been determined, this may readily be translated into a tensor-based notation if desired (with this process being summarised in Appendix B).

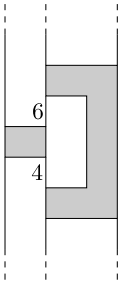


FIG. 2. Diagram showing part of a tensor network in which two indices, labelled 4 and 6, connect the same two tensors. Dotted lines represent summed indices which connect the two tensors shown here to the rest of the network.

Note that when two or more consecutive indices in an index sequence connect the same tensors, for example the indices labelled 4 and 6 in Fig. 2, it is assumed that these indices should first be combined into a single index having dimension equal to the product of its constituents. For example, if the index labelled 4 in Fig. 2 can range in value from 1 to 2 (i.e. the index labelled 4 is of dimension 2) and the index labelled 6 is of dimension 3, these two indices may be replaced by a single index t ranging from 1 to $2\times 3 = 6$, as per Table I. Note that this combining of indices applies even when composite objects are involved, and thus if a contraction sequence for Fig. 1(iv) begins `9 4 6 ...`, the first contraction (over index 9) combines tensors C and E, generating a new tensor which then shares both indices 4 and 6 with tensor D. Indices 4 and 6 are therefore to be combined into a single index before contracting.

Taking this choice of notation into account, a pseudocode function `getcost(`$\mathcal{N}_\mathrm{Z}$`)` which returns both the minimum cost associated with contracting a network $\mathcal{N}_\mathrm{Z}$ and an index-based contraction sequence realising that cost may be represented as follows:

Define: `[sequence cost] = getcost(`$\mathcal{N}_\mathrm{Z}$`)`

1: Let the value of `cost` be marked as unassigned.

2: Iterating over all bipartitions of $\mathcal{N}_\mathrm{Z}$, for each bipartition:

2a: Let $\mathcal{N}_\mathrm{X}$ and $\mathcal{N}_\mathrm{Y}$ denote the sub-networks resulting from this bipartition, and X and Y denote the tensors obtained on contracting $\mathcal{N}_\mathrm{X}$ and $\mathcal{N}_\mathrm{Y}$.

2b: Let `cost1` be the cost of contracting tensor X with tensor Y.

2c: Let `seq1` be the list of index labels connecting tensor X with tensor Y.

2c: If $\mathcal{N}_\mathrm{X}$ contains more than one tensor, let `[seqX costX]=getcost(`$\mathcal{N}_\mathrm{X}$`)`, else let `seqX` be empty and let `costX=0`.

2d: If $\mathcal{N}_\mathrm{Y}$ contains more than one tensor, let `[seqY costY]=getcost(`$\mathcal{N}_\mathrm{Y}$`)`, else let `seqY` be empty and let `costY=0`.

2e: Let `costZ` be `cost1+costX+costY`.

2f: Let `seqZ` be the sequential concatenation `[seqX seqY seq1]`.

2g: If the value of `cost` is unassigned, or if a value has been assigned to `cost` and this value satisfies `cost>costZ`, let `cost` take the value of `costZ` and let `sequence` take the value of `seqZ`.

3: Return the values of `sequence` and `cost`.

TABLE I. Example replacement of the indices labelled 4 and 6 (of dimensions 2 and 3 respectively) with a single index, denoted t, of dimension 6.

Value of index t	Associated values of indices 4 and 6	
	Index 4	Index 6
1	1	1
2	1	2
3	1	3
4	2	1
5	2	2
6	2	3

2. *Iteration over non-disjoint sub-networks*

Given sufficient time, the algorithm described above will always return the minimum cost for contracting any tensor network $\mathcal{N}_\mathrm{Z}$ and an associated contraction sequence. However, some thought must be given with respect to the manner in which the algorithm explores the space of bipartitions of the network, or else "sufficient time" may be very long indeed. In the limit of a fully-connected network of n tensors (i.e. one in which every tensor is connected to every other tensor), the number of bipartitions to be considered in the initial step scales as $2^{n-1}-1$ and the number of admissible contraction sequences scales as

$$\frac{n!(n-1)!}{2^{n-1}}. \tag{2}$$

In practice tensor networks of interest are less interconnected than this, so restriction to bipartitions yielding non-disjoint sub-networks will substantially reduce these figures, but the cost of identifying an optimal contraction sequence may still be anticipated to rise rapidly with n.

When the requirement for non-disjoint subnetworks is relaxed in Sec. II B, the number of possible contraction sequences for a tensor network returns to the worst case scenario given in Eq. (2). However, it turns out that only a fraction of these sequences need be taken into account in order to be certain of obtaining a sequence of minimum cost. It is therefore useful to describe a procedure for iterating over all bipartitions of a non-disjoint tensor network $\mathcal{N}_\mathrm{Z}$ which yield non-disjoint sub-networks $\mathcal{N}_\mathrm{X}$ and $\mathcal{N}_\mathrm{Y}$, with this procedure requiring only minimal modification in Sec. II B 3.

To begin, consider a fully-connected network of n tensors. For such a network, one may explore all possible bipartitions yielding non-disjoint sub-networks using the following procedure: First, assign a unique label to each tensor (e.g. for $n = 7$, the letters A... G). Write out these labels in sequence. Then to iterate over bipartitions, let a variable x count from 1 to $2^{n-1} - 1$. For each value of x write its binary representation under the list of tensor labels. Where a label is associated with a 0, this tensor is placed in group one. Where a label is associated with a 1, this tensor is placed in group two. This process is illustrated in Table II for $n = 7$. The value $x = 0$ may be omitted as for this value group 2 is empty, so this does not represent a bipartition of the network. It is also unnecessary to consider $x \in [2^{n-1} + 1, 2^n]$ as the resulting bipartitions are equivalent to those for $x \in [0, 2^{n-1}]$ with groups one and two interchanged.

For a realistic network with fewer interconnections, however, following this procedure to bipartition the network will frequently result in more than two disconnected pieces. Iteration over all possible bipartitions as per Table II is therefore relatively inefficient for realistic tensor networks, but may be built on to realise a more economical iteration over bipartitions of the tensor network as follows:

First, note that if a bipartition such as Group 1 = [ABCD], Group 2 = [EFG] represents a valid bipartition of the original network (i.e. that neither group is disjoint), then it is unnecessary to consider also Group 1 = [EFG], Group 2 = [ABCD]. We may choose to impose this constraint by constructing an algorithm which only generates bipartitions in which tensor A is in group two.

Next, define Tier 1 (denoted $\mathcal{T}_1$) as being a set of tensors whose only member is tensor A, and let tensor A be in group two. Now define Tier 2 as the set of all tensors in contact with a member of Tier 1 which is in group two, but which are not themselves in Tier 1, and let all members of Tier 2 initially belong to group one. This definition is then extended to higher tiers such that Tier x for $x > 1$ is the set of all tensors which are connected to a tensor in $\mathcal{T}_{x-1}$ which belongs to group two, but which are not themselves a member of tier $\mathcal{T}_y$ for any $y < x$. Initially, $\mathcal{T}_x$ will be empty for all $x > 2$. In general, not all tensors will be assigned to a tier, and tensors not assigned to a tier are always taken to belong to group one. This initial configuration therefore places tensor A in group two, and all other tensors in group one.

TABLE II. Using the binary representation of a counter x to iterate over all possible bipartitions of a list of seven tensors denoted A... G.

x	A	B	C	D	E	F	G	Group 1	Group 2
1	0	0	0	0	0	0	1	ABCDEF	G
2	0	0	0	0	0	1	0	ABCDEG	F
3	0	0	0	0	0	1	1	ABCDE	FG
4	0	0	0	0	1	0	0	ABCDFG	E
				...					
63	0	1	1	1	1	1	1	A	BCDEFG

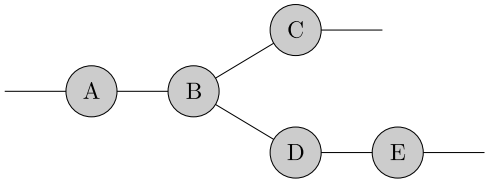


FIG. 3. Example tensor network used to demonstrate the contiguous-group iteration algorithm described in Sec. II A 1. The output of this algorithm is given in Table III.

Advancing to the next bipartition configuration is now performed as follows:

1. Let x be the number of the highest non-empty tier (initially, Tier 2).
2. In the manner of Table II, perform a binary incrementation of the group assignations in Tier x. For example, in Fig. 1(iv) Tier 2 would comprise tensors C, D, and F. Initially, all three belong to group one. After incrementation, one tensor (say F) would now belong to group two.
3. If incrementing the configuration resulted in binary overflow, such that after the increment all elements of Tier x belong to group one (e.g. for Tier x containing three elements, this would occur on incrementing from 111 to 000), then:
   - 3a. If the tier which overflowed was Tier 1, iteration is complete. Stop.
   - 3b. Otherwise, let Tier x be empty, and return to Step 1.
4. Having changed the group assignations in Tier x, Tier $x + 1$ may now be non-empty. Let all tensors in Tier $x + 1$ be in group one.
5. Read off new configuration.

Repeating this process iterates over all bipartitions for which group two is non-disjoint. It is still necessary to check whether group one is non-empty and non-disjoint, but by using this approach, the number of bipartitions which must be examined may be substantially reduced. A full example is given in Fig. 3 and Table III.

B. Disjoint networks

Discussion thus far has been restricted to tensor networks which are non-disjoint. However, sometimes constructing an optimal contraction sequence for a tensor network—even one which is non-disjoint—may necessarily involve taking the product of two tensors which do not share any common indices (an "outer product"), with an example of such a network being seen in Fig. 4, and to

Iteration	Tier 1	Tier 2	Tier 3	Tier 4	No Tier	Group 1	Group 2	Group 1 non-disjoint and non-empty?
1	Tensors: A Groups: 2	Tensors: B Groups: 1	Tensors: - Groups: -	Tensors: - Groups: -	Tensors: C,D,E Groups: 1,1,1	B,C,D,E	A	Yes
2	Tensors: A Groups: 2	Tensors: B Groups: 2	Tensors: C,D Groups: 1,1	Tensors: - Groups: -	Tensors: E Groups: 1	C,D,E	A,B	No
3	Tensors: A Groups: 2	Tensors: B Groups: 2	Tensors: C,D Groups: 1,2	Tensors: E Groups: 1	Tensors: - Groups: -	C,E	A,B,D	No
4	Tensors: A Groups: 2	Tensors: B Groups: 2	Tensors: C,D Groups: 1,2	Tensors: E Groups: 2	Tensors: - Groups: -	C	A,B,D,E	Yes
5	Tensors: A Groups: 2	Tensors: B Groups: 2	Tensors: C,D Groups: 2,1	Tensors: - Groups: -	Tensors: E Groups: 1	D,E	A,B,C	Yes
6	Tensors: A Groups: 2	Tensors: B Groups: 2	Tensors: C,D Groups: 2,2	Tensors: E Groups: 1	Tensors: - Groups: -	E	A,B,C,D	Yes
7	Tensors: A Groups: 2	Tensors: B Groups: 2	Tensors: C,D Groups: 2,2	Tensors: E Groups: 2	Tensors: - Groups: -	-	A,B,C,D,E	No

TABLE III. Iteration over all bipartitions of a tensor network for which group two is not disjoint, using the algorithm described in Sec. II C. The tensor network being iterated over is given in Fig. 3.

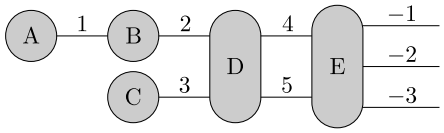


FIG. 4. An example tensor network for which the optimal contraction sequence involves performing an outer product. All indices are of dimension χ, with $\chi > 1$.

locate such a contraction sequence it is necessary to consider bipartitions which generate disjoint sub-networks. The problems associated with this are two-fold: First, it is necessary to extend the index-based contraction sequence notation employed in Sec. II A 1 to encompass outer products, and second, it is necessary to identify an optimised exploration of the space of contraction sequences which includes disjoint sub-networks when required, but which does not involve evaluating the full list of $O(n!)$ possible contraction sequences described in Sec. II A 2.

Initially considering only networks $\mathcal{N}_Z$ which are non-disjoint, Sec. II B 1 examines the conditions under which it is necessary to consider outer products in order to be certain of finding a sequence of optimal cost, and Sec. II B 2 introduces an extension of the index-labelling notation of Sec. II A 1 which is capable of describing the resulting contraction sequences. It is shown in Sec. II B 1 that even when outer products (and hence bipartitions which yield disjoint sub-networks) are taken into account, it is still not necessary to examine all bipartitions of a network $\mathcal{N}_Z$, and Sec. II B 3 then explains how the iteration algorithm of Sec. II A 2 may be modified to include the necessary additional bipartitions without needing to iterate over *all* bipartitions of a non-disjoint network $\mathcal{N}_Z$. Finally, Sec. II B 4 extends this approach to situations where the original tensor network $\mathcal{N}_Z$ is itself disjoint.

1. *Optimal sequences involving outer products*

As mentioned in Sec. II A 2, the performance of any algorithm which attempts to iterate over *all* bipartitions of a tensor network, including bipartitions yielding disjoint sub-networks, will scale very poorly with the number of tensors involved. Fortunately, it is not necessary to consider all bipartitions—only those which are necessary in order to be certain of finding an optimal contraction sequence—and it is consequently possible to derive a number of restrictions on which bipartitions yielding disjoint subnetworks should be taken into consideration.

First, note that performing an outer product between two tensors is formally equivalent to contracting over a shared index of dimension 1. Now consider a contraction sequence in which an outer product is taken between two tensors, A and B, and then the resulting object (AB) is contracted with a further tensor C which shares one or more indices with A but none with B. This situation is illustrated in Fig. 5, in which

- all indices on tensor A not connecting to tensor C have been combined into a single index of dimension a,
- all indices on tensor C not connecting to tensor A have been combined into a single index of dimension c,
- all indices connecting tensors A and C have been combined into a single index of dimension d,
- and all indices on tensor B have been combined into a single index of dimension b.

As the combination procedure (which is illustrated in Ta-

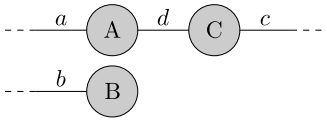


FIG. 5. An example sub-network whose contraction involves an outer product. Should tensor A be contracted with tensor C before or after performing an outer product involving tensor B? Letters $a, \ldots, d$ denote index dimensions.

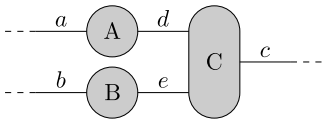


FIG. 6. Two tensors A and B share no common index; should they be combined using an outer product before contracting with tensor C? Letters $a, \ldots, e$ denote index dimensions.

ble I) is reversible, this may be done without loss of generality. Tensors A and B share no common indices by definition, as the contraction of A with B is required to be an outer product. Given this labelling, one can ask under what circumstances the contraction sequence ((AB)C) may be superior to the sequence ((AC)B). Evaluating the costs of these two contraction sequences yields the inequality

$$abd + abcd < acd + abc \tag{3}$$

which, for a, b, c, and d being positive integers, is satisfiable only if $b = 1$ or $d = 1$. Defining the *total dimension* of a tensor as the product of the dimensions of all its indices, it follows that performing an outer product between any two tensors A and B may only ever potentially be advantageous if:

- the next contraction to be performed is also an outer product (equivalent to $d = 1$),
- the next contraction is with a tensor C that shares non-trivial indices (i.e. indices of dimension greater than 1) with both A and B, or
- one or more of tensors A and B has total dimension 1 (i.e. is a single number).

Networks involving tensors of total dimension 1 represent a special case of limited interest, and may always be factored out as a simple numerical multiplier. It is convenient to neglect these cases in the interest of obtaining a better-performing algorithm for tensor networks of practical interest.

Next, consider the most general situation in which tensor C shares non-trivial indices with both A and B. This situation may be represented as shown in Fig. 6, where $d \geq 2$ and $e \geq 2$. All indices on tensor A not connecting to tensor C have been combined into a single index of dimension a, and similarly for b and c. All indices common to tensors A and C have been combined into a single index d, and all those common to B and C have been combined into a single index e. As before, tensors A and B share no common indices by definition. The three possible contraction sequences for this diagram and their associated costs are now given in Table IV, and requiring that ((AB)C) be cheaper than both of the other two (permitting an outer product to potentially offer an advantageous contraction sequence) gives rise to the inequalities

$$c(b + d - bd) > bd \tag{4}$$

$$c(a + e - ae) > ae. \tag{5}$$

Since all indices must take positive integer values greater than or equal to one, and d and e are required to be greater than or equal to two, it is seen that these equations admit solutions only for

$$a = b = 1, \tag{6}$$

reducing to

$$c > d \tag{7}$$

$$c > e. \tag{8}$$

Consequently, if the optimal contraction of tensors A, B, and C must begin with an outer product between A and B, tensors A and B may not have any non-trivial indices other than those which they share with tensor C, and tensor C must have at least one such index, of dimension satisfying Eqs. (7) and (8).

In combination with the results inferred from Fig. 5 it follows that sequences involving an outer product on n tensors $\mathrm{A}_1, \ldots, \mathrm{A}_n$ need only be considered when they satisfy the following criteria:

- The outer products are performed pairwise (see Appendix A).
- After performing $n-2$ pairwise outer products, the series of outer products yields a pair of tensors A and B each having one or more non-trivial indices shared with tensor C and no non-trivial indices not shared with tensor C.

As it is never optimal to perform an outer product between two tensors which share a non-trivial summed index (as opposed to summing over that index), and as networks involving tensors of trivial total dimension have been excluded, it follows that *all* the tensors $\mathrm{A}_1, \ldots, \mathrm{A}_n$ each have one or more non-trivial indices shared with tensor C and no non-trivial indices not shared with tensor C.

TABLE IV. Contraction sequences and associated costs for the tensor network shown in Fig. 6.

Sequence	Cost
((AB)C)	$abde + abcde$
((AC)B)	$acde + abce$
((BC)A)	$bcde + abcd$

Next, consider the construction of tensor C. It turns out that Eqs. (6)–(8) permit us to conclude that a contraction sequence for tensor C can always be chosen such that it is not itself the result of an outer product, while still obtaining an optimal cost overall: If

- the optimal final contraction in the construction of C is unavoidably an outer product of two tensors $\tilde{\mathrm{A}}$ and $\tilde{\mathrm{B}}$,
- the optimal contraction of A, B, and C begins with the outer product of A and B, and
- the product of contracting A with B is denoted $\tilde{\mathrm{C}}$,

then in addition to Eqs. (6)–(8) the indices of tensors $\tilde{\mathrm{A}}$, $\tilde{\mathrm{B}}$, and $\tilde{\mathrm{C}}$ must also satisfy the constraints of Eqs. (6)–(8), implying

$$c = 1 \tag{9}$$

$$ab > d \tag{10}$$

$$ab > e. \tag{11}$$

Simultaneous satisfaction of Eqs. (6)–(8) and Eqs. (9)–(11) is impossible, implying that in the construction of an optimal contraction sequence it is never necessary to contract together two tensors both produced using an outer product. Consequently, when exploring bipartitions of a tensor network $\mathcal{N}_{\mathrm{Z}}$ it is still permissible to impose that one of the two groups of tensors generated by a bipartition be non-disjoint. If the other is disjoint, it is further permissible to discard this bipartition if the disjoint group possesses any unpaired indices other than those which are to be contracted with the non-disjoint group.

Finally, when a sequence of outer products is to be performed on a set of tensors $\mathrm{A}_1, \ldots, \mathrm{A}_n$, the optimal order of contraction may be determined by associating with each tensor a total dimension ξ_i, and proceeding as follows:

1. Identify the two tensors having the smallest total dimensions.
2. Remove those two tensors from the list and replace with their outer product.
3. Repeat until only one tensor remains.

For example, having four tensors $\mathrm{A}_1, \ldots, \mathrm{A}_4$ of total dimensions $\xi_1 = \chi^4$, $\xi_2 = \chi^5$, $\xi_3 = \chi^6$, $\xi_4 = \chi^8$ for some value χ, an optimal sequence for taking the outer product of these tensors is $((\mathrm{A}_1\mathrm{A}_2)(\mathrm{A}_3\mathrm{A}_4))$. The resulting tensor is then contracted with tensor C.

2. Notation

Addressing first the problem of notation, consider again the example given in Fig. 4. Using a tensor-based notation, for this example the optimal contraction sequence may be represented

$$((((\mathrm{AB})\mathrm{C})\mathrm{D})\mathrm{E}). \tag{12}$$

Tensors (AB) and C share no common indices, and thus the contraction of (AB) with C is an outer product. In index-based notation the order over which the labelled indices are to be contracted may be written

```
1 2 3 4 5
```
(13)

This sequence, however, makes no reference to the outer product which must be performed, and attempting to reconstruct Eq. (12) will erroneously yield the suboptimal contraction sequence ((((AB)D)C)E). Admittedly the sequence of Eq. (13) is not unique; recall that when multiple indices connect the same two tensors, they will be combined into a single index before contraction and hence their ordering is immaterial. However, the alternatives fare no better. Like Eq. (13) the index sequence `1 2 3 5 4` implies a tensor contraction sequence ((((AB)D)C)E), and the other two alternatives—`1 3 2 4 5` and `1 3 2 5 4`—fare no better, implying the sequence (((AB)(CD))E) which is again sub-optimal. Some extension of index-based sequence notation is clearly required.

In Sec. II B 1 it was shown that to be certain of identifying a contraction sequence of minimum cost, one need only take into account sequences where all tensors A, B, etc. participating in an outer product share a summed index with some other tensor X which does not participate in the outer product, and the next operation is to contract the result of the outer product with X. The presence of an outer product of n tensors may therefore be denoted by inserting $n-1$ zeros into the contraction sequence. Conversely, when interpreting a contraction sequence, on encountering a string of $n-1$ zeros the response is then as follows:

1. Read the next x indices of the contraction sequence, where x is the maximum value such that n tensors will be involved in summing over all x indices.
2. Identify the n tensors involved.
3. One of these tensors will be connected to all the others; this is tensor X as described above.
4. Perform an optimal pairwise outer product on all tensors except X.
5. Contract the resulting object with tensor X.

This process is usefully supplemented by two additional sub-algorithms. The first is the process described in Sec. II B 1 for determining the optimal ordering of a sequence of outer products, and the second identifies X:

1. Consider the first of the x indices. This index will connect two tensors, to be denoted A and B.
2. Advance through the x indices until one is found which appears on A or B, but not both. The tensor (either A or B) on which this index appears is tensor X.

Provided index-based sequence notation is only used to describe outer products of the sort discussed in Sec. II B 1, it follows that any contraction sequence consistent with these restrictions may be described using the

zero notation described above. The bipartitioning procedure employed by the Netcon algorithm may be chosen so as to yield all contraction sequences which are compliant with these restrictions, and only contraction sequences which are compliant with these restrictions (see Sec. II B 3), and since there always exists an optimal contraction sequence which is consistent with these restrictions (see Sec. II B 1), it follows that Netcon will always find an optimal sequence, and that the optimal contraction sequence returned by Netcon may always be described using the zero notation. This notation is slightly less intuitive than the pairwise bracketing notation used in (for example) Eq. (12), but has the advantage of being a linear rather than heirarchically-structured representation of the contraction sequence. For easy reference the interpretation of an index-based contraction sequence which may include the zero notation is summarised in Appendix B.

For the example given in Fig. 4, the index sequence encoding the optimal tensor contraction sequence ((((AB)C)D)E) is

```
1 0 2 3 4 5.
```

3. *Iteration over disjoint and non-disjoint sub-networks of a non-disjoint network*

Given the results of Sec. II B 1, when searching for the optimal contraction sequence for a non-disjoint network $\mathcal{N}_\mathrm{Z}$ it is only necessary to iterate over all bipartitions such that at least one of the resulting groups is non-disjoint. This may be achieved as follows:

Let there be an additional Tier not discussed in Sec. II A 2, named Tier 0, and initially empty. All tensors in Tier 0 are always in group one. Now select a tensor A to use as Tier 1, and perform the iteration algorithm described in Sec. II A 2, but do not discard bipartitions in which group one is disjoint unless it also possesses open indices not shared with group two. Let $\mathcal{N}_\mathrm{X}$ denote the sub-network which is allowed to be disjoint, and replace step 2c of the iteration algorithm with

2c: If $\mathcal{N}_\mathrm{X}$ contains only one tensor, let `seqX` be empty and let `costX=0`, otherwise:

- 2c1: Let n be the number of disjoint sections making up sub-network $\mathcal{N}_\mathrm{X}$, and let $\mathcal{N}_{\mathrm{X}_1},\ldots,\mathcal{N}_{\mathrm{X}_n}$ enumerate these sections.
- 2c2: Let `costOP` be the cost for taking the outer product of these disjoint sections according to the method of Sec. II B 1.
- 2c3: For each disjoint section $\mathcal{N}_{\mathrm{X}_i}$, let `[seqX`$_i$ `costX`$_i$`]=getcost(`$\mathcal{N}_{\mathrm{X}_i}$`)`.
- 2c4: Let `seqX` be the concatenation `[seqX`$_1$ `seqX`$_2$ `... seqX`$_n$`]`.
- 2c5: Let `costX` be `costOP` $+\sum_i$ `costX`$_i$.
- 2c6: Prepend $n-1$ zeros onto `seq1`.

Completion of the iteration algorithm of Sec. II A 2, amended as above, corresponds to iterating over:

- all bipartitions for which both groups are non-disjoint, and
- all bipartitions in which the group containing tensor A is non-disjoint but the other group is disjoint.

Next, move tensor A from Tier 1 to Tier 0 and select a different tensor (e.g. B) to form a new Tier 1. Repeat the iteration algorithm of Sec. II A 2, but this time examine only bipartitions for which group one is disjoint (partitions for which both groups are non-disjoint have already been examined during the iteration in which Tier 0 was empty). On completion the contents of Tier 1 are again added to Tier 0 and a new Tier 1 is selected. This procedure repeats until all tensors are in Tier 0. These additional steps yield

- all bipartitions in which the group containing tensor A is disjoint but the other group is non-disjoint,

completing iteration through all necessary bipartitions of $\mathcal{N}_\mathrm{Z}$.

4. *Computing the optimal contraction sequence for a disjoint network*

Up to this point, it has been assumed that the original tensor network $\mathcal{N}_\mathrm{Z}$ supplied to the Netcon algorithm is non-disjoint. While iterating through bipartitions of $\mathcal{N}_\mathrm{Z}$ the function `getcost()` may yield disconnected sub-networks $\mathcal{N}_\mathrm{X}$, but each of these is then further separated into its non-disjoint components and `getcost()` is recursively applied to each non-disjoint component in turn. Extension to situations where the original network $\mathcal{N}_\mathrm{Z}$ is disjoint is not difficult.

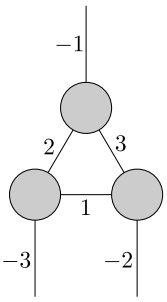
Consider again the argument presented in Sec. II B 1 regarding the timing of outer products. As it is always possible to defer an outer product until such a time as the next contraction involves indices shared with both elements of the outer product, it follows that for a disjoint network $\mathcal{N}_\mathrm{Z}$, taking the outer product between the various elements of $\mathcal{N}_\mathrm{Z}$ should be the final step in contracting the tensor network. The process of determining an optimal contraction for a network $\mathcal{N}_\mathrm{Z}$ composed of n disjoint sub-networks may therefore be broken down into that of determining an optimal contraction sequence for each disjoint sub-network, concatenating these sequences, and then appending $n-1$ zeros onto the end of the sequence to indicate the performance of $n-1$ outer products between the remaining n tensors. The interpretation of a string of zeros in the contraction sequence is therefore as follows:

- When a string of $n-1$ zeros is encountered, this indicates that $n-1$ outer products are to be performed.

- If only n tensors remain, then the outer product of all n tensors is to be evaluated pairwise in the optimal order according to the procedure given in Sec. II B 1.
- If more than n tensors remain, the n tensors to be contracted together are to be determined according to the algorithm given in Sec. II B 2, as is tensor X. The outer product of these n tensors is then to be evaluated pairwise in the optimal order according to the procedure given in Sec. II B 1.

Note that some caution is required if the tensor network $\mathcal{N}_\mathrm{Z}$ includes indices of dimension 1, since (as noted in Sec. II B 1) contracting over a shared index of dimension 1 is formally equivalent to performing an outer product between two tensors, and should be treated as such. This is particularly important where a tensor network $\mathcal{N}_\mathrm{Z}$ is disjoint, for example being made up of two disjoint sub-networks $\mathcal{N}_\mathrm{X}$ and $\mathcal{N}_\mathrm{Y}$, and one of these sub-networks, say $\mathcal{N}_\mathrm{X}$, is made up of a further pair of sub-networks, $\mathcal{N}_\mathrm{V}$ and $\mathcal{N}_\mathrm{W}$, connected only by indices of dimension 1. Writing V, W, X, and Y for the tensors formed on contracting sub-networks $\mathcal{N}_\mathrm{V}$, $\mathcal{N}_\mathrm{W}$, $\mathcal{N}_\mathrm{X}$, and $\mathcal{N}_\mathrm{Y}$ respectively, if sub-networks $\mathcal{N}_\mathrm{X}$ and $\mathcal{N}_\mathrm{Y}$ are each contracted independently and then an outer product is performed between tensors X and Y, an optimal sequence for the contraction of sub-network $\mathcal{N}_\mathrm{X}$ may be identified in which the final pairwise product is an outer product between tensors V and W. The final steps in the contraction of network $\mathcal{N}_\mathrm{Z}$ are then necessarily ((VW)Y). However, it is possible that the optimal sequence for contracting tensors V, W, and Y is in fact either ((VY)W) or ((WY)V), and thus the optimal sequence for network $\mathcal{N}_\mathrm{Z}$ may be missed. To ensure identification of the optimal contraction sequence for $\mathcal{N}_\mathrm{Z}$ it is necessary to identify *all* sub-networks which are effectively disjoint, in this case $\mathcal{N}_\mathrm{V}$, $\mathcal{N}_\mathrm{W}$, and $\mathcal{N}_\mathrm{Y}$, independently contract each of these sub-networks, then determine the optimal contraction sequence for the resulting objects (which are permitted to be of trivial total dimension) according to the algorithm given in Sec. II B 1. The identification of all effectively disjoint sub-networks is easily achieved by stripping from the network all indices of dimension 1. An optimal contraction sequence for the original network may then be achieved by computing an optimal sequence for the reduced network and appending to this sequence all labels which correspond to indices of trivial dimension.

In this approach, contraction over trivial indices is deferred until contraction of network $\mathcal{N}_\mathrm{Z}$ is otherwise complete, at which point they represent traces over indices of dimension 1. Performing such a trace incurs zero computational cost. In general it is unwise to leave indices uncontracted—if a pair of tensors A and B share a common index of dimension d and that index is left unsummed when evaluating (AB), the effect is to unnecessarily increase the total dimension of (AB) by a factor of d^2, and similarly increase the cost of any subsequent contractions involving tensor (AB). When $d = 1$, however, no additional cost is associated with this deferral.



Index label	Dimension
1	6χ
2	χ
3	χ^2
−1	6χ
−2	χ
−3	χ^2

FIG. 7. A simple network for which the optimal contraction sequence varies with the value of χ.

It is further noted that when $\mathcal{N}_\mathrm{Z}$ is non-disjoint, stripping of trivial indices in this manner may cause the network to separate into two or more disjoint components. Although this process is not obligatory for non-disjoint $\mathcal{N}_\mathrm{Z}$ if the Netcon algorithm is to return an optimal contraction sequence it is harmless, and if the stripping causes $\mathcal{N}_\mathrm{Z}$ to separate into disjoint components then the ability to exploit this disjoint structure yields a substantial performance increase for the Netcon algorithm. Consequently, the default behaviour of the Netcon algorithm is to ignore the existence (and thus defer contraction) of trivial indices even for non-disjoint networks $\mathcal{N}_\mathrm{Z}$.

C. Efficient exploration of network bipartitions

1. *Excluding high-cost contraction sequences*

When exploring all possible bipartitions of a network $\mathcal{N}_\mathrm{Z}$, a substantial performance increase may be obtained by excluding those leading to higher-cost contraction sequences. How this is best done depends upon the manner in which the dimensions of the tensor indices are to be specified, with two different approaches being considered in this paper:

1. In the first approach, index dimensions are specified as integer values. The returned contraction sequence then minimises the cost for these specific values.
2. In the second approach, index dimensions are specified in the form $a\chi^b$ for some parameter χ, and the sequence returned is one which is optimal in the limit of large χ. A cost of the form $a\chi^{b+1}$ is therefore always larger than a cost of $a'\chi^b$ regardless of the values taken by a and a'.

The contraction sequences returned by these approaches will not necessarily always coincide. For example, consider the simple network shown in Fig. 7. Contracting this network according to the index sequence `3 2 1` incurs a cost of $72\chi^6$, while the sequence `1 2 3` incurs a cost of $12\chi^7$. The sequence `3 2 1` is therefore clearly to be preferred in the limit of large χ, but sequence `1 2 3` is

preferred for explicit index costs corresponding to values of $\chi < 6$.

Consider first the task of excluding higher-cost contraction sequences when index dimensions are specified as integers. Let ξ_{cap} denote the upper bound on contraction sequences to be explored, and re-examine the `getcost(`$\mathcal{N}_{\text{Z}}$`)` algorithm described in Sec. II A 1. In step 2, the algorithm iterates over bipartitions of network $\mathcal{N}_{\text{Z}}$ into sub-networks $\mathcal{N}_{\text{X}}$ and $\mathcal{N}_{\text{Y}}$, whose contractions yield tensors X and Y. The minimal contraction cost associated with each such bipartition is then the cost of combining tensors X and Y, `cost1`, added to the minimal costs associated with contracting each of the sub-networks $\mathcal{N}_{\text{X}}$ and $\mathcal{N}_{\text{Y}}$ (costs `costX` and `costY` respectively). Evaluation of these latter costs proceeds by recursion and may be lengthy if the number of tensors in either of these sub-networks is large. However, if the value of `cost1` exceeds ξ_{cap} then it is unnecessary to proceed with the evaluation of either `costX` or `costY` as the minimal cost associated with this bipartition, `cost1` + `costX` + `costY`, also necessarily exceeds ξ_{cap}. Similarly, if `cost1` is acceptable but `cost1` + `costX` exceeds ξ_{cap} then evaluation of `costY` is unnecessary.

When evaluating the minimal cost to contract a tensor network $\mathcal{N}_{\text{Z}}$, it is entirely possible that no contraction sequence will be found which is compatible with the upper bound on cost imposed by ξ_{cap}. Furthermore, this situation may be encountered at any level of recursion (for example, the cost associated with a particular bipartition of $\mathcal{N}_{\text{Z}}$ may be acceptable, but there may not exist any contraction sequences for $\mathcal{N}_{\text{X}}$ which are cheaper than ξ_{cap}). In this situation the `getcost(`$\mathcal{N}_{\text{Z}}$`)` algorithm should return a flag indicating failure to find an acceptable contraction sequence, and also the smallest cost for which further investigation of a contraction sequence was discarded. This value will be denoted ξ_{fail} and corresponds to the smallest of the following:

1. The smallest value of `cost1` which exceeded ξ_{cap}.
2. The smallest value of `cost1+failX` where `failX` is the returned value of ξ_{fail} for sub-network $\mathcal{N}_{\text{X}}$.
3. The smallest value of `cost1+costX` which exceeded ξ_{cap}.
4. The smallest value of `cost1+costX+failY` where `failY` is the returned value of ξ_{fail} for sub-network $\mathcal{N}_{\text{Y}}$.
5. The smallest value of `cost1+costX+costY` which exceeded ξ_{cap}.

This value, ξ_{fail}, then corresponds to the minimum value to which ξ_{cap} must be increased in order to permit further evaluation of some of the discarded contraction sequences. Conversely it is also possible that a contraction sequence will be found for a cost less than ξ_{cap}, in which situation ξ_{cap} should be decreased accordingly as more expensive sequences need no longer be considered.

Next, consider the task of excluding higher-cost contraction sequences when index dimensions are specified in the form $a\chi^b$. For large χ, a cost scaling as $\text{O}(\chi^n)$ will always be greater than one scaling as $\text{O}(\chi^{n-1})$, and a simple means of placing an upper bound on the cost of contraction sequences to be investigated is to impose an upper limit on admissible powers of χ. Since the total cost of a sequence is just the sum of the costs of its constitutent pairwise contractions, this constraint may in turn be imposed by placing an upper limit ξ_{powercap} on the maximum acceptable power of χ to appear in the cost of a single pairwise tensor contraction. (Also note that individual pairwise contractions are always monomial in χ.) If the cost `cost1` associated with a given bipartition of a tensor network $\mathcal{N}_{\text{Z}}$ into sub-networks $\mathcal{N}_{\text{X}}$ and $\mathcal{N}_{\text{Y}}$ involves a power of χ greater than ξ_{pcap} then the costs associated with sub-networks $\mathcal{N}_{\text{X}}$ and $\mathcal{N}_{\text{Y}}$ need not be evaluated, and `getcost()` can advance to the next candidate bipartition.

A similar condition again applies to the cost associated with contracting sub-network $\mathcal{N}_{\text{X}}$, such that if `costX` contains a power of χ greater than some value ξ_{pcap} then evaluation of the cost to contract sub-network $\mathcal{N}_{\text{Y}}$ is unnecessary. However, in contrast with the situation where index dimensions are specified as integer values, it is no longer necessary to explicitly check this also for `cost1` + `costX` as if `cost1` is of order at most $\text{O}(\chi^{\xi_{\text{pcap}}})$ and `costX` is of order at most $\text{O}(\chi^{\xi_{\text{pcap}}})$ then the same is necessarily true of `cost1` + `costX`. In the event that no contraction sequence exists which has a cost of at most $\text{O}(\chi^{\xi_{\text{pcap}}})$, `getcost()` should return a value ξ_{fail} indicating the minimum value to which ξ_{pcap} must be increased to permit the further evaluation of some discarded contraction sequences. In this approach, the value of ξ_{fail} is given by the smallest of the following:

1. The power of χ appearing in the smallest value of `cost1` which exceeds $\text{O}(\chi^{\xi_{\text{pcap}}})$.
2. The smallest value of ξ_{fail} returned by a call to `getcost(`$\mathcal{N}_{\text{X}}$`)`.
3. The smallest value of ξ_{fail} returned by a call to `getcost(`$\mathcal{N}_{\text{Y}}$`)`.

Finally, if a sequence is identified having a cost of $\text{O}(\chi^{\xi})$ where $\xi < \xi_{\text{pcap}}$ then the value of ξ_{pcap} should be decreased accordingly.

Regardless of whether index dimensions are specified as integers or in the form $a\chi^b$, imposing an appropriate cap on the contraction sequences to be evaluated can result in a significant performance increase. However, choosing an appropriate value for ξ_{cap} or ξ_{pcap} may be problematic if the cost of the optimal contraction sequence is not known in advance. Setting this value too high may result in suboptimal performance of the Netcon algorithm, and setting it too low will result in failure to identify an acceptable contraction sequence. A resolution to this problem is presented in Sec. II C 2.

2. Caching of sub-network outcomes

During the recursive bipartitioning of a tensor network $\mathcal{N}_\mathrm{Z}$ described in Sec. II A 1, the function `getcost()` will frequently be called upon to evaluate a sub-network which has been encountered before. [For example, one might first partition Fig. 1(iv) into $\mathcal{N}_\mathrm{ABCFG}$ and $\mathcal{N}_\mathrm{DE}$, or one might split it into $\mathcal{N}_\mathrm{BCG}$ and $\mathcal{N}_\mathrm{ADEF}$ then split the latter into $\mathcal{N}_\mathrm{AF}$ and $\mathcal{N}_\mathrm{DE}$. Both approaches involve evaluating the cost associated with contracting sub-network $\mathcal{N}_\mathrm{DE}$.] A significant performance increase may be obtained by caching results for previously-solved sub-networks, such that on the second and later encounters their optimal cost and a sequence yielding this cost may be looked up rather than being calculated by invoking `getcost()` for the sub-network.

Even greater advantage may be obtained when the caching of costs for sub-networks is combined with the exclusion of high-cost contraction sequences as described in Sec. II C 1. If the value of ξ_cap (or ξ_pcap, as appropriate) is initially set to a value known to be too low to permit successful contraction of the network, `getcost()` will fail and will return the lowest-valued cost for which further evaluation was aborted. If ξ_cap is then increased to this value, `getcost()` is called again, *and the cached results from the last attempt are retained*, with this procedure being repeated until a successful contraction sequence is found, then the net effect is to automatically exclude from evaluation all sequences involving a single contraction cost greater than some threshold ξ_cutoff, where ξ_cutoff corresponds to the largest single contraction cost arising during the optimal contraction sequence. Where more than one optimal contraction sequence exists, let i enumerate these sequences and let ξ^i_max be the maximum cost associated with a single contraction in sequence i. The value of ξ_cutoff is then the smallest value in the set $\{\xi^i_\mathrm{max}\}$.

In addition to caching the costs and sequences for successfully-evaluated sub-networks, when employing a cutoff ξ_cap it is also useful to cache the costs at which evaluation for a given sub-network is aborted (ξ_fail). Thus if evaluation of sub-network $\mathcal{N}_\mathrm{P}$ is aborted due to a minimum step cost of at least $p > \xi_\mathrm{cap}$ (i.e. the cheapest decomposition of $\mathcal{N}_\mathrm{P}$ has been determined to inevitably involve a contraction having a cost of at least p) and sub-network $\mathcal{N}_\mathrm{Q}$ is similarly aborted due to a minimum cost $q > p$, then if ξ_cap is increased to p it is immediately apparent that sub-network $\mathcal{N}_\mathrm{P}$ should be re-evaluated as it may perhaps now be contractible with all steps costing less than ξ_cap, whereas there is no need yet to further investigate sub-network $\mathcal{N}_\mathrm{Q}$. If re-evaluation of sub-network $\mathcal{N}_\mathrm{P}$ shows that it can indeed be contracted with a maximum cost-per-step of ξ_cap then the corresponding cost and sequence are added to the cache. If not—if it turns out that after performing the step of cost p a more expensive contraction is also required—then the cached value of ξ_fail for $\mathcal{N}_\mathrm{P}$ should be updated instead.

Using this approach, it is interesting to compare the order with which previously-excluded sub-networks become available for further investigation when index dimensions are specified as integers, versus when they are specified in the form $a\chi^b$. For instance, consider a situation where index dimensions are specified in the form $a\chi^b$ and initial investigation of two sub-networks $\mathcal{N}_\mathrm{P}$ and $\mathcal{N}_\mathrm{Q}$ shows that contraction of $\mathcal{N}_\mathrm{P}$ will involve at least one step costing χ^6, while contraction of $\mathcal{N}_\mathrm{Q}$ will involve at least one step costing $2\chi^6$. Under this approach, both sub-networks will become eligible for further investigation at the same time (when ξ_pcap is increased to 6). Whether $\mathcal{N}_\mathrm{P}$ or $\mathcal{N}_\mathrm{Q}$ is evaluated first will depend upon which is encountered first when iterating through bipartitions of the parent network. If, however, an explicit value is substituted for χ, say $\chi = 5$, and the Netcon algorithm is invoked for the same network but with the index dimensions specified as integers, the increase of ξ_cap is such that bipartitions and sub-networks are investigated in strict order of cost. Sub-network $\mathcal{N}_\mathrm{P}$ will be investigated first, when ξ_cap rises to 5^6, and $\mathcal{N}_\mathrm{Q}$ will only be investigated if no solution exists for a cost less than 2×5^6, with this taking place on a later iteration of the Netcon algorithm once ξ_cap has been raised to 2×5^6. In practice, the former approach strikes a better balance between making candidate sequences available more rapidly (and so reducing the number of iterations of the Netcon algorithm before a solution is found) and the exclusion of large numbers of unnecessarily costly sequences which may cause individual iterations of the Netcon algorithm to take an excessively long time. Consequently, where it is appropriate to do so, specification of index dimensions in the form $a\chi^b$ is to be preferred. In the reference implementation provided with this paper, the slower performance when index dimensions are specified as integers has been offset by requiring that any time ξ_cap is incremented, the new value is at least equal to the old value of ξ_cap multiplied by the dimension of the smallest non-trivial index of the tensor network.

D. Traces

It is worth making brief mention of tensor networks involving traces. Unlike tensor contractions, which require multiplication, traces may be evaluated using only addition. As this operation is relatively cheap, it is assumed that the costs associated with tracing are negligible when compared to those associated with pairwise contraction. Consequently, the Netcon algorithm performs all traces on a network before any pairwise contractions. Further, as noted in Sec. II B 4, when two tensors share multiple indices of non-trivial dimension it is never advantageous to leave one or more of these indices uncontracted while combining the tensors. Consequently, the only traces ever appearing in the optimal sequences returned by the Netcon algorithm are those where the initial network contains traces, which are performed at the beginning of the contraction sequence, and those over indices of dimen-

sion 1, which are performed at the end of the contraction sequence as discussed in Sec. II B 4,

III. REFERENCE IMPLEMENTATION

An example implementation of the Netcon algorithm described in Sec. II, written in MATLAB and C++, may be obtained as follows: While viewing the abstract page for this paper on arXiv.org, select "Download > Other formats", then "Download source". On expanding the resulting archive, the files comprising an implementation of the Netcon algorithm are `netcon.m` and `netcon_cpp.cpp`. The implementation requires MATLAB 2011a or above and a compatible C++ compiler, and has been tested under MATLAB 2011a with both Microsoft Visual C++ 2010 and Apple XCode 4.5.2.

A. Compilation

The reference implementation of the Netcon algorithm comprises a MATLAB function `netcon()` which may be invoked from the MATLAB command line. It requires compilation of a C++ component before use, which is achieved using the MATLAB `mex` command. First, ensure that a supported C++ compiler has been configured using the command `mex -setup` as described in the MATLAB documentation. Then download and extract the contents of `netcon.zip` to obtain the files `netcon.m` and `netcon_cpp.cpp`. The C++ component of the implementation may then be compiled using the command

```
netcon compile
```

which is equivalent to invoking `mex netcon_cpp.cpp`.

Note that by default the C++ portion of the code makes use of automatic memory allocation and deallocation for variable-length arrays, and that this feature is not supported by some C++ compilers, including Microsoft Visual C++ 2010. If compilation using the default configuration fails, an alternative configuration which employs manual memory allocation and deallocation may be compiled using the command

```
netcon compile safe
```

(equivalent to `mex -DSAFE netcon_cpp.cpp`). This configuration, however, is associated on some systems with a performance decrease by a factor of 2-3, and thus when supported the former configuration is preferred.

B. Invocation

Invocation of the algorithm is via the MATLAB command

```
[sequence cost] = netcon(legLinks,verbosity,
          costType,xiCap,legCosts,keepTriv);
```

and takes between one and six input parameters, as follows:

`legLinks`: This parameter describes the tensor network for which an optimal contraction sequence is sought. To construct `legLinks`, first draw the tensor network using the customary graphical notation (summarised in §1.2 of Ref. 45), with each tensor being represented by a shape, each summed index by a line connecting the two tensors on which it appears, and each unsummed index by a line with one free end and the other end attached to the tensor on which it appears [e.g. Fig. 1(i)]. Next, label each summed index with a unique positive integer, and each unsummed index with a unique negative integer, descending consecutively from -1 [e.g. Fig. 1(iv)]. For each tensor T, now construct a $1 \times n_T$ matrix where n_T is the number of indices attached to tensor T, with entries corresponding to the labels associated with those indices. Ordering of the indices is unimportant. If there are m tensors, then there are m such matrices, which may be denoted $M_i,\ i \in \{1 \dots m\}$. Finally, `legLinks` comprises a $1 \times m$ cell array, with the m entries of this array corresponding to the matrices M_i, with ordering once again being unimportant. For example, the labelling given in Fig. 1(iv) may be associated with the input parameter

```
legLinks = {[-1 1 2 3],[2 4 5 6],[1 5 7 -3],
            [3 8 4 9],[6 9 7 10],[-2 8 11 12],
            [10 11 12 -4]}.
```

`verbosity`: Determines the level of output generated by `netcon()`. For `verbosity` = 0, an optimal index contraction sequence is returned in `sequence` and the associated cost is returned in `cost` but operation is otherwise silent. For `verbosity` = 1, a message is generated every time the upper bound on tensor contraction costs is increased as described in Sec. II C 1, and on completion an optimal sequence and the associated cost are displayed on screen. For `verbosity` = 2, behaviour is as for `verbosity` = 1 but candidate contraction sequences and associated costs are announced when these sequences constitute the lowest-cost contraction sequence found so far. The final sequence and cost announced then correspond to the optimal solution reported at `verbosity` = 1 and returned in the output variables `sequence` and `cost`. Default value: 2.

`costType`: To determine an optimal contraction sequence associated with a given tensor network, it is necessary to specify the dimension of each index in the network. Index dimensions may either be specified as integers, or in the form $a\chi^b$, where χ is an unspecified parameter which is presumed to be large. The former is indicated by `costType` = 1, and the latter by `costType` = 2. Default value: 2. Note that when using `costType` = 2, indices of fixed dimension d (such as the physical indices of an MPS or PEPS) may be represented by setting $a = d$ and $b = 0$.

`xiCap`: When searching for an optimal contraction sequence, `netcon()` initially restricts itself to sequences

having a cost of at most $\xi_{\rm cap}$ (for `costType` = 1) or $O(\chi^{\xi_{\rm pcap}})$ (for `costType` = 2), where `xiCap` represents $\xi_{\rm cap}$ or $\xi_{\rm pcap}$ respectively. The value of `xiCap` will automatically increase if no contraction sequence exists which satisfies this constraint. Setting `xiCap` too high can incur extremely large overheads, whereas the process of automatic increase is relatively low-cost due to the caching of data described in Sec. II C 2. It is therefore recommended that `xiCap` be left at its default value of 1 unless the cost to contract the network is already known.

`legCosts`: The format of this input parameter is dependent upon the value of `costType`.

1. For `costType` = 1, `legCosts` is an $\ell \times 2$ matrix whose first column consists of index labels and whose second column gives the dimensions associated with those labels. If `legCosts` is not specified, it is assumed that each index has dimension 2.
2. For `costType` = 2, `legCosts` comprises an $\ell \times 3$ matrix where ℓ is the total number of unique index labels. Each row then comprises three entries, $[x\ a\ b]$, where x is a index label (and hence a positive or negative integer), and a and b specify the dimension of index x in terms of the cost parameter χ, such that $\dim(x) = a\chi^b$. If `legCosts` is not specified, it is assumed that each index (whether summed or unsummed) has dimension χ. Note that b may take the value zero, permitting a fixed cost to be specified for some indices.

Regardless of the value of `costType`, if `legCosts` is specified, each index label must appear in the first column precisely once. Note that tensors of total dimension 1 are not supported.

`keepTriv`: Setting this value to `true` suppresses the stripping of trivial indices. Setting `keepTriv` to `true` is not recommended as it may result in poorer performance of the Netcon algorithm, and the return of sub-optimal contraction sequences for some disjoint tensor networks. The only advantage to enabling `keepTriv` is that it may occasionally replace outer products with contraction over one or more trivial indices. There is no cost benefit to this replacement, but the resulting sequence may look a little neater. Default value: `false`.

On completion, `netcon()` returns an optimal contraction sequence and associated cost. These are specified as follows:

`sequence`: Sequence over which the indices of the tensor network should be summed in order to contract the network for minimum cost. For the interpretation of this sequence, see Appendix B.

`cost`: Specifies the total number of multiplication operations associated with optimal contraction of the tensor network, for example according to the sequence returned in `sequence`. For `costType` = 2 this value is a number. For `costType` = 1 the cost takes the form of a polynomial in χ,

$$\sum_{i=0}^{\chi_{\max}} a_i \chi^i, \tag{14}$$

and this is returned as a $1 \times \chi_{\max}$ array whose entries `cost(i)` correspond to the coefficients a_{i-1}.

Note that when a tensor network involves one or more traces, these may always be evaluated before network contraction begins. Evaluating a trace involves only addition operations, not multiplication, and thus these operations are relatively cheap and are ignored when computing the value of `cost` (though the presence of costs associated with tracing over indices will be noted in the text output if `verbosity` > 0).

C. Sample invocation and output

As a simple example, consider the tensor network given in Fig. 1. Allowing `costType` and `legCosts` to take their default values, corresponding to each index having dimension χ, netcon may be invoked with verbosity level 1 by the command

```
[sequence cost]=netcon({[-1 1 2 3],[2 4 5 6],
                        [1 5 7 -3],[3 8 4 9],
                        [6 9 7 10],[-2 8 11 12],
                        [10 11 12 -4]},1);
```

returning the output

```
Looking for solutions of cost O(X^1)
Looking for solutions of cost O(X^6)
Looking for solutions of cost O(X^7)
Looking for solutions of cost O(X^8)

Best sequence:   9 4 6 5 7 2 1 3 11 12 8 10
Cost:            2X^8 + 2X^7 + 2X^6 + 0X^5
                 + 0X^4 + 0X^3 + 0X^2 + 0X^1
                 + 0X^0
```

indicating that the cost of contracting this network scales as $O(\chi^8)$, and that for any given value of χ the actual cost of performing this contraction will be $2\chi^8+2\chi^7+2\chi^6$. The sequence and cost are also returned in the variables `sequence` and `cost`:

```
sequence = [9 4 6 5 7 2 1 3 11 12 8 10]
cost     = [0 0 0 0 0 0 2 2 2].
```

IV. PERFORMANCE

The aim of the Netcon algorithm is to present a practical, automated means of evaluating the optimal contraction for a tensor network, as an alternative to human evaluation. In order to present a compelling argument

for this algorithm, it is necessary to show that an implementation of the algorithm is capable of delivering results within a timeframe comparable to or better than that of human evaluation. It can also be argued that Netcon would be attractive even if slightly slower than the human, as it would free the human from the necessity of performing what is a painstaking and frequently tedious process while also eliminating the risk of human error.

In practice the reference implementation `netcon()` supplied with this paper has proven substantially faster than human evaluation for all tensor networks examined in this paper, with the additional benefit of providing absolute certainty that no more efficient contraction sequence remains to be found. (For more complicated tensor networks, performing an exhaustive search is not an option for a human, and the problem is often instead reduced to that of finding the contraction sequence of lowest cost possible within a reasonable timeframe.) Performance data are provided in Table V.

Of particular interest is the result for the 4:1 2D MERA described in Refs. 6 and 43. Variational optimisation of this Ansatz involves the contraction of eleven distinct tensor networks (and a further forty-nine which are equivalent to reflections or rotations of these eleven), and represents a particularly challenging task for Netcon given both the large number of tensors involved (27 in each diagram) and the large number of interconnections between the tensors. Determining optimal contraction sequences for each of these networks using `netcon()` takes times ranging from one hundred and thirty-one minutes to four days, although the eleven different diagrams can all be attacked simultaneously by running multiple instances of MATLAB across several machines. This might seem to be an inconveniently lengthy computation simply to determine an optimal contraction sequence, but the advantage of running Netcon becomes clear when it is noticed that the contraction sequences returned by `netcon()` have total costs of $\mathrm{O}(\chi^{26})$, whereas the cost of optimising this Ansatz is reported in Ref. 7 to be of $\mathrm{O}(\chi^{28})$. Assuming that the value reported in Ref. 7 is not a typo, the contraction sequences located by the Netcon algorithm are therefore significantly more efficient than any previously identified by human evaluation.

For simpler networks, it is clear from Table V that `netcon()` offers a substantial advantage in both speed and convenience when it comes to determining optimal contraction sequences. For more complex networks, the challenge presented both to `netcon()` and to human operators increases, and the ability of `netcon()` to perform an exhaustive search within a practical timeframe continues to make it the approach of choice.

TABLE V. Performance comparison between `netcon()` and a human operator determining optimal contraction sequences for a selection of tensor networks. For the TEBD algorithm, the time quoted is that taken to find the optimal contraction sequence for application of an imaginary time evolution update [see Figs. 3(i)-(ii) of Ref. 39], and has physical dimension 2 and bond dimension D. For TTNs it is assumed that the Hamiltonian is made up of nearest-neighbour terms on a square lattice, and the time is for finding an optimal contraction sequence to perform energy minimisation on one tensor in the tree. For each MERA algorithm listed, the times given are for the network in that algorithm taking the longest time to evaluate using `netcon()`. The times t_{human} correspond to the time for the author to independently identify a contraction sequence having the same cost to leading order as that given by `netcon()`. All calculations were performed using the reference implementation of `netcon()` with `costType` = 2 and with the initial value of `xiCap` set to 1, running in MATLAB R2011a on a Macbook Pro with a 2.2GHz Intel Core i7 processor and 8Gb of 1333MHz RAM. The C++ portion of the software was compiled using Apple XCode 4.5.2.

Tensor network	Number of tensors	Optimal cost	t_{human}	t_{netcon}
TEBD	6	$\mathrm{O}(D^3)$	49s	0.04s
3:1 1D TTN	5	$\mathrm{O}(\chi^6)$	10s	0.04s
9:1 2D TTN	8	$\mathrm{O}(\chi^{14})$	35s	0.06s
3:1 1D MERA	7	$\mathrm{O}(\chi^8)$	61s	0.06s
2:1 1D MERA	11	$\mathrm{O}(\chi^9)$	19m	0.12s
9:1 2D MERA	20	$\mathrm{O}(\chi^{16})$	38m	21s
4:1 2D MERA	27	$\mathrm{O}(\chi^{26})$	−[a]	4 days

[a] Human analysis of this network failed to yield a contraction sequence cheaper than $\mathrm{O}(\chi^{28})$. See Sec. IV of main text for discussion.

V. DISCUSSION

This paper has presented a detailed description of the Netcon algorithm for determining an optimal sequence for the contraction of a tensor network, and the associated cost, along with a reference implementation in MATLAB called `netcon()`. It has been shown that this reference implementation has a performance which is competitive in real-world applications, evaluating optimal contraction sequences substantially faster than a human researcher. By automating this time-consuming and error-prone task, the Netcon algorithm is capable of substantially facilitating the development and implementation of more advanced tensor network algorithms and Ansätze.

Authors who use Netcon, `netcon()`, or any derivative works, are requested to cite this paper in resulting publications. An example citation might read "Our results were computed using an implementation of our variational Ansatz in Fortran95, with contraction sequences determined by the Netcon algorithm [1]" where "[1]" is a citation of this paper.

The author thanks the Ontario Ministry of Research and Innovation ERA for financial support.

Appendix A: Preferential nature of pairwise contractions

In Sec. II A 1 it was stated that an optimal contraction sequence for a tensor network may always be realised as a series of pairwise contractions. To see this, consider the contraction of three tensors, A, B, and C, to yield a single tensor D. Let ξ_{ab} denote the product of the dimensions of all indices on tensor A which connect to tensor B, and similarly for ξ_{ac} and ξ_{bc}. Let ξ_a denote the product of all indices on tensor A which do not connect to either B or C, and similarly for ξ_b and ξ_c. The dimension of a set containing no indices, is always one. Contracting these three tensors as a single process involves a cost of

$$2\xi_a\xi_b\xi_c\xi_{ab}\xi_{ac}\xi_{bc}. \tag{A1}$$

For example, for the contraction

$$D^\alpha_\epsilon = A^{\alpha\beta\gamma}B_{\beta\delta}C^\delta_{\gamma\epsilon}, \tag{A2}$$

we have

$$\begin{aligned} \xi_a &= \dim\alpha, \\ \xi_b &= 1, \\ \xi_c &= \dim\epsilon, \\ \xi_{ab} &= \dim\beta, \\ \xi_{ac} &= \dim\gamma, \\ \text{and } \xi_{bc} &= \dim\delta. \end{aligned} \tag{A3}$$

For each element of D^α_ϵ it is necessary to sum over $\xi_{ab}\xi_{ac}\xi_{bc}$ different contributions (corresponding to the enumeration of indices β, γ, and δ), each involving two multiplications, and there are then $\xi_a\xi_b\xi_c$ entries in D^α_ϵ, for the total number of multiplication operations given in Eq. (A1). In contrast, pairwise contraction may be achieved by any of the sequences ((AB)C), ((AC)B), or ((BC)A), where ((XY)Z) means "contract tensor X with tensor Y, then contract the result with tensor Z". The sequence ((AB)C) is readily seen to attract a total cost of

$$\xi_a\xi_b\xi_{ab}\xi_{ac}\xi_{bc} + \xi_a\xi_b\xi_c\xi_{ac}\xi_{bc} \tag{A4}$$

multiplication operations, with those for ((AC)B) and ((BC)A) being achieved by the relevant label permutations. Since all parameters in Eqs. (A1) and (A4) take a value greater than or equal to one, the value of Eq. (A4) is always less than or equal to that of Eq. (A1). The argument extends directly to contraction of four or more tensors, with the cost of sequential pairwise contraction continuing to always be less than or equal to that of more complicated contractions, and consequently for any tensor network it is always possible to identify a minimum-cost contraction sequence in which all contractions proceed in a pairwise fashion. Note that no assumption has been made about the values of the index dimensions, and thus this result holds even for contraction sequences involving outer products, which may be equated with contraction over indices of dimension 1.

Appendix B: Interpretation of contraction sequences specified as a list of indices

The Netcon algorithm described in this paper takes as its input a description of a tensor network where each summed index is associated with a positive integer label and each open index is associated with a negative integer label. As its output the algorithm returns an optimal contraction sequence for the specified tensor network, specified as a list of positive integer labels possibly interspersed with zeros, and the cost of performing this contraction (corresponding to the number of multiplication operations required). Interpretation of tensor contraction sequences specified in this form is described in the paper above, but is summarised in this Appendix for convenience.

Starting with a list of tensors in the network to be contracted, and beginning with the first index of the sequence, contraction of a tensor network proceeds as follows:

1: If the sequence list is empty, stop. Contraction of the tensor network is complete.

2: Read the first entry, i_1.

3: If $i_1 = 0$:

- 3a: Read a further $x-1$ entries, for a total of x entries, denoted $i_1, \ldots, i_x$, where x is the largest possible value such that all x entries are zero.
- 3b: Let n be the number of tensors currently in the list of tensors. If $x = n - 1$:
  - 3b1: Using the outer product algorithm given below, perform an outer product of all n remaining tensors. Denote the result of this outer product X. Delete all n tensors from the list. Add tensor X to the list.
  - 3b2: Delete entries $i_1, \ldots, i_x$ from the sequence.
  - 3b3: Go to step 1.
- 3c: Otherwise (i.e. $x \neq n - 1$):
  - 3c1: Delete entries $i_1, \ldots, i_x$ from the sequence.
  - 3c2: Read the next y entries from the sequence (denoted $j_1, \ldots, j_y$), and list all tensors on which indices $j_1, \ldots, j_y$ appear, where y is the largest possible value such that all entries $j_1, \ldots, j_y$ are nonzero and the number of tensors in the list is precisely $x + 2$.
  - 3c3: Let these tensors be referred to as $\mathrm{A}_1, \ldots, \mathrm{A}_{x+2}$.
  - 3c4: Let B_1 and B_2 denote the tensors on which index j_1 appears. Identify the smallest value of z such that index j_z appears on a tensor which is neither B_1 nor B_2. Index j_z also appears on either B_1 or B_2. Let X denote this tensor (either B_1 or

B_2). Note that tensor X will be a member of the list $A_1, \ldots, A_{x+2}$.

3c5: Using the outer product algorithm given below, perform an outer product of all tensors $A_1, \ldots, A_{x+2}$ except for tensor X. Let the result of this outer product be denoted Y.

3c6: Indices $j_1, \ldots, j_y$ all appear on both X and Y. Evaluate the product of tensors X and Y, denoted (XY), summing over all possible configurations of the indices $j_1, \ldots, j_y$.

3c7: Delete tensors $A_1, \ldots, A_{x+2}$ from the list of tensors. Add tensor (XY) to the list of tensors.

3c8: Delete indices $j_1, \ldots, j_y$ from the sequence.

3c9: Return to step 1.

4: Otherwise (i.e. $i_1 \neq 0$), identify which tensors index i_1 appears on.

5: If index i_1 appears on only one tensor, it represents a trace. Trace over index i_1, delete this index from the sequence, and return to step 1.

6: Otherwise: Index i_1 appears on two tensors, A and B. Read a further $x-1$ indices, for a total of x indices, denoted $i_1, \ldots, i_x$, where x is the largest possible value such that all indices $i_1, \ldots, i_x$ appear on both tensor A and tensor B.

7: Evaluate the product of tensors A and B, denoted (AB), summing over all possible configurations of the indices $i_1, \ldots, i_x$.

8: Delete tensors A and B from the list of tensors. Add tensor (AB) to the list of tensors.

9: Delete indices $i_1, \ldots, i_x$ from the sequence.

10: Return to step 1.

When called upon to perform an outer product of m tensors, this should be done by applying the following simple algorithm:

1: Define the total dimension of a tensor as the product of the dimensions of its indices.
2: List the m participating tensors.
3: Identify the two tensors having the smallest total dimensions.
4: Remove those two tensors from the list.
5: Add their outer product to the list.
6: Repeat steps 3-5 until only one tensor remains.

[1] A. Feiguin, S. Trebst, A. W. W. Ludwig, M. Troyer, A. Kitaev, Z. Wang, and M. H. Freedman, Phys. Rev. Lett. **98**, 160409 (2007).
[2] S. Trebst, E. Ardonne, A. Feiguin, D. A. Huse, A. W. W. Ludwig, and M. Troyer, Phys. Rev. Lett. **101**, 050401 (2008).
[3] R. König and E. Bilgin, Phys. Rev. B **82**, 125118 (2010).
[4] R. N. C. Pfeifer, G. Evenbly, and G. Vidal, Phys. Rev. A **79**, 040301 (2009).
[5] R. N. C. Pfeifer, P. Corboz, O. Buerschaper, M. Aguado, M. Troyer, and G. Vidal, Phys. Rev. B **82**, 115126 (2010).
[6] L. Cincio, J. Dziarmaga, and M. M. Rams, Phys. Rev. Lett. **100**, 240603 (2008).
[7] G. Evenbly and G. Vidal, Phys. Rev. Lett. **102**, 180406 (2009).
[8] G. Evenbly, R. N. C. Pfeifer, V. Picó, S. Iblisdir, L. Tagliacozzo, I. P. McCulloch, and G. Vidal, Phys. Rev. B **82**, 161107 (2010).
[9] R. M. Noack and S. R. White, Phys. Rev. B **47**, 9243 (1993).
[10] M. Fannes, B. Nachtergaele, and R. F. Werner, J. Stat. Phys. **66**, 939 (1992).
[11] B. Friedman, J. Phys.: Condens. Matter **9**, 9021 (1997).
[12] H. Otsuka, Phys. Rev. B **53**, 14004 (1996).
[13] L. Tagliacozzo, G. Evenbly, and G. Vidal, Phys. Rev. B **80**, 235127 (2009).
[14] V. Murg, F. Verstraete, and J. I. Cirac, Phys. Rev. A **75**, 033605 (2007).
[15] V. Murg, F. Verstraete, and J. I. Cirac, Phys. Rev. B **79**, 195119 (2009).
[16] D. Muth, B. Schmidt, and M. Fleischhauer, New J. Phys. **12**, 083065 (2010).
[17] V. Murg, F. Verstraete, O. Legeza, and R. M. Noack, Phys. Rev. B **82**, 205105 (2010).
[18] S. Montangero, M. Rizzi, V. Giovannetti, and R. Fazio, Phys. Rev. B **80**, 113103 (2009).
[19] P. Corboz, G. Evenbly, F. Verstraete, and G. Vidal, Phys. Rev. A **81**, 010303 (2010).
[20] P. Corboz, R. Orús, B. Bauer, and G. Vidal, Phys. Rev. B **81**, 165104 (2010).
[21] P. Corboz, M. Lajkó, A. M. Läuchli, K. Penc, and F. Mila, Phys. Rev. X **2**, 041013 (2012).
[22] P. Corboz, S. R. White, G. Vidal, and M. Troyer, Phys. Rev. B **84**, 041108 (2011).
[23] S. Yan, D. A. Huse, and S. R. White, Science **332**, 1173 (2011).
[24] S. Depenbrock, I. P. McCulloch, and U. Schollwöck, Phys. Rev. Lett. **109**, 067201 (2012).
[25] G. Vidal, Phys. Rev. Lett. **99**, 220405 (2007).
[26] G. Vidal, Phys. Rev. Lett. **101**, 110501 (2008).
[27] G. Evenbly and G. Vidal, Phys. Rev. B **79**, 144108 (2009).
[28] G. Vidal, in *Understanding Quantum Phase Transitions*, edited by L. D. Carr (Taylor & Francis, Boca Raton, 2010).
[29] B. Swingle, Phys. Rev. D **86**, 065007 (2012).
[30] B. Swingle, arXiv:1209.3304v1 [hep-th] (2012).
[31] S. Singh and G. Vidal, arXiv:1303.6716v2 [cond-mat.str-el] (2013).
[32] X. Chen, Z.-C. Gu, and X.-G. Wen, Phys. Rev. B **83**, 035107 (2011).

[33] X. Chen, Z.-C. Gu, and X.-G. Wen, Phys. Rev. B **84**, 235128 (2011).
[34] N. Schuch, D. Pérez-García, and I. Cirac, Phys. Rev. B **84**, 165139 (2011).
[35] L. Cincio and G. Vidal, Phys. Rev. Lett. **110**, 067208 (2013).
[36] S. R. White, Phys. Rev. Lett. **69**, 2863 (1992).
[37] U. Schollwöck, *January 2011 Special Issue*, Ann. Phys. **326**, 96 (2011).
[38] G. Vidal, Phys. Rev. Lett. **93**, 040502 (2004).
[39] G. Vidal, Phys. Rev. Lett. **98**, 070201 (2007).
[40] Y.-Y. Shi, L.-M. Duan, and G. Vidal, Phys. Rev. A **74**, 022320 (2006).
[41] F. Verstraete and J. I. Cirac, arXiv:cond-mat/0407066v1 (2004).
[42] J. Jordan, R. Orús, G. Vidal, F. Verstraete, and J. I. Cirac, Phys. Rev. Lett. **101**, 250602 (2008).
[43] G. Evenbly and G. Vidal, arXiv:0710.0692v2 [quant-ph] (2007).
[44] M. Aguado and G. Vidal, Phys. Rev. Lett. **100**, 070404 (2008).
[45] R. N. C. Pfeifer, *Simulation of Anyons Using Symmetric Tensor Network Algorithms*, Ph.D. thesis, The University of Queensland (2011), arXiv:1202.1522v2 [cond-mat.str-el].